Gremlin RAM

Table of Contents

[The kinds of RAM 3](#_Toc308339596)

[BRAM 3](#_Toc308339597)

[Actual Gremlin BRAM Configuration 4](#_Toc308339598)

[SDRAM 7](#_Toc308339599)

[Device Registers 7](#_Toc308339600)

[BG0\_MAP\_ADDRESS (no bits) 7](#_Toc308339601)

[BG0\_IMAGE\_ADDRESS (no bits) 7](#_Toc308339602)

[BG0\_GLOBAL\_PALETTE P 7](#_Toc308339603)

[BG0\_VISIBLE V 7](#_Toc308339604)

[BG0\_SCROLL YYYXXX 7](#_Toc308339605)

[BG1\_MAP\_ADDRESS (no bits) 7](#_Toc308339606)

[BG1\_IMAGE\_ADDRESS (no bits) 7](#_Toc308339607)

[BG1\_GLOBAL\_PALETTE P 7](#_Toc308339608)

[BG1\_VISIBLE V 7](#_Toc308339609)

[BG1\_SCROLL YYYXXX 7](#_Toc308339610)

[BG2\_MAP\_ADDRESS (no bits) 7](#_Toc308339611)

[BG2\_IMAGE\_ADDRESS (no bits) 7](#_Toc308339612)

[BG2\_GLOBAL\_PALETTE P 8](#_Toc308339613)

[BG2\_VISIBLE V 8](#_Toc308339614)

[BG2\_SCROLL YYYXXX 8](#_Toc308339615)

[SPRITE\_DATA\_REGISTER[256] IIIIIIPP PPPPRRRY YYYYYYYX XXXXXXXX 8](#_Toc308339616)

[SPRITE\_POOL\_SELECT S 8](#_Toc308339617)

# The kinds of RAM

There are two kinds of RAM in the system:

1. BRAM – This is the Xilinx internal 32K of BRAM, or “Block Ram”, configured as u8’s on the CPU side. This is incredibly fast, internal memory inside the FPGA. It’s dual-ported as well. Every operation against this memory will complete in 1 cycle, making it perfect for VRAM. It’s also broken into 16 independent banks within the Xilinx, so it can be configured so that several different systems can access different BRAM’s at once, and thus all operate simultaneously and independently.
2. SDRAM – This is an external 8MB of SDRAM, configured as u16’s. SDRAM is dynamic, meaning that it must take time-outs once in a while, to perform a “refresh” – keeping the data alive so that it doesn’t fade away. This means that read and write timing for this kind of memory is rather unpredictable. You may make a read request, and have to wait for an extra 16+ cycles for an internal refresh operation to complete, as compared with the best-case read, which can be as low as 1 cycle if reading sequentially from the same page of memory.

## BRAM

The BRAM has the following attributes:

* 32Kbytes made of 16 BRAMs in any of the following configurations:
  + 16K x 1 (2-color fonts)
    - 256 x 8x8 cells
    - 16384 pixels
  + 8K x 2 (4-color)
    - 128 x 8x8 cells
    - 8192 pixels
  + 4K x 4 (16-color)
    - 64 x 8x8 cells
    - 4096 pixels
  + 2K x 8 (byte-based)
    - 32 x 8x8 cells
    - 2048 pixels
    - 64 x 32 cell map x 2 BRAMS = 64 x 64 map
    - 8bits applied to various bit-depths
      * 64 x 16-color cells requires 6 bits (2 bits for per-cell palette selection)
      * 128 x 4-color cells requires 7 bits (1 bit for per-cell palette selection)
      * 256 x 2-color cells requires 8 bits (no per-cell palette selection)
  + 1K x 16 (good for palettes or scanline buffers)
    - 1024 palette entries
    - 4 x 256 color palettes
    - 64 x 16 color palettes
    - 3 x 320 pixels wide scanlines
    - 2 x 400 pixels wide scanlines
  + 512 x 32 (good for getting a large amount of data per single fetch)
    - Sprite Registers (enough for 2 sets of 256 sprites – double buffering!)
* Example configurations:
  + 400x300
    - 64x64 map with 64 x 16-color cells and 4 palettes available per cell: 4 BRAMs
    - 64x64 map with 128 x 4-color cells and 2 palettes available per cell: 4 BRAMs
    - 64x64 map with 256 x 2-color cells and 1 palette available per cell: 4 BRAMs
    - 64x64 map with 128 x 16-color cells and 2 palettes available per cell: 5 BRAMs
    - 64x64 map with 256 x 16-color cells and 1 palette available per cell: 7 BRAMs
    - 400x300 framebuffer with 4-colors per pixel: 16 BRAMs
    - 400x300 framebuffer with 2-colors per pixel: 9 BRAMs
  + 320x240
    - 64x32 map with 128 x 16-color cells and 2 palettes available per cell: 4 BRAMs
    - 64x32 map with 256 x 4-color cells and 1 palette available per cell: 4 BRAMs
    - 64x32 map with 256 x 2-color cells and 1 palette available per cell: 3 BRAMs
    - 64x32 map with 196 x 16-color cells and 1 palette available per cell: 5 BRAMs
    - 64x32 map with 256 x 16-color cells and 1 palette available per cell: 6 BRAMs
    - 64x32 map with 128 x 256-color cells and 2 palettes available per cell: 6 BRAMs
* Guaranteed 1 cycle access time @ the BRAM’s clock speed.
* Dual Ported. CPU, DMA, Blitter, etc on one side, Video Pixel-generator on the other side.

### Actual Gremlin BRAM Configuration

#### VGA Clock Management

Clocks available to compute next scanline = 832 per scanline x 2 scanlines = 1664 clocks

1664 – 320 clocks per visible pixel = 1344 clocks available for rendering sprites.

1344 – 256 clocks to find visible sprites = 1088 clocks to paint visible sprites.

1088 / 16 pixels per row of a sprite = 68 sprites per scanline before I run out of clocks.

…Or (with sprites inserted between cell layers)…

1664 – 320 \* 2 clocks per visible pixel = 1024 clocks available for rendering sprites.

1024 – 256 clocks to find visible sprites = 768 clocks to paint visible sprites.

768 / 16 pixels per row of a sprite = 48 sprites per scanline before I run out of clocks.

#### BRAM Allocations

##### Scanline Buffer (1 BRAM)

* 1 x Scanline Buffer (SCANLINE\_BUFFER\_VRAM)
  + BRAM is organized as:
    - PortA: Scanline Renderer – 1024 x 18bits, configured as (6bit paletteId | 4bit colorIndex | 8 bits unused), high address bit chooses one of two scanlines (one is for displaying, one is for generating, and they are swapped on every new vertical pixel).
    - PortB: PixelBufferVgaSignalGenerator –sees the same configuration as PortA.

NOTE: The Scanline Buffer is not accessable by the CPU. It is used exclusively by the VGA image generation circuitry.

##### Sprites (5 BRAMs)

* 1 x Sprite Registers (SPRITE\_REGS\_VRAM)
  + BRAM is organized as:
    - PortA: 512 x 32bits
      * Each address is a sprite data register
    - PortB: 2K x 8bits
      * For CPU direct access.
* 4 x Sprite Pixels @ 4bpp = 4096 x 4 = 16384 pixels = 64 16x16 sprites (SPRITE\_IMAGE\_VRAM)
  + BRAM is organized as:
    - PortA: 4 BRAMs x 4K x 4bits for VGA engine to render sprite data.
    - PortB: 4 BRAMs x 2K x 8bits for CPU direct access.

##### BG0 (3 BRAMs) – Good for fonts

* 1 x BG Map (BG0\_MAP\_VRAM)
* 2 x BG Cells = 128 cells (4bit) with 2 palettes available per cell (BG0\_CELL\_VRAM)

##### BG1 (3 BRAMs) – Good for Gameplay FG

* 1 x BG Map (BG1\_MAP\_VRAM)
* 2 x BG Cells = 128 cells (4bit) with 2 palettes available per cell (BG1\_CELL\_VRAM)

##### BG2 (3 BRAMs) – Good for Gameplay BG

* 1 x BG Map (BG2\_MAP\_VRAM)
* 2 x BG Cells = 128 cells (4bit) with 2 palettes available per cell (BG2\_CELL\_VRAM)

##### Palettes (1 BRAM)

* 1 x Palette RAM (PALETTE\_VRAM)
* 64 x 4bit palettes
  + First 4 are accessable by BG
  + All 64 are accessable by Sprites
* BRAM is organized as:
  + PortA: VGA Engine - 1024 x 16bit colors, where the top 4 bits are always 0.
  + PortB: CPU Bus - 512 x 32bit address space, where each address contains 2 x 12-bit colors, each padded to 16bits.

0 BG0\_0 BG1\_00 BG2\_00 SPRITE

1 BG0\_1 SPRITE

2 BG1\_01 SPRITE

3 SPRITE

4 BG1\_10 SPRITE

5 SPRITE

6 BG1\_11 SPRITE

7 SPRITE

8 BG2\_01

…

63 SPRITE

BG0PALETTE 00000P

BG1PALETTE 000PC0

BG2PALETTE 0PC000

SPRITEPALETTE PPPPPP

G = Global palette index

C = Cell palette index

#### Scanline Rendering Engine

##### Paint BG’s

For(x=0;x<320;x++)

Bg0Index = BG0\_MAP\_VRAM[] -> BG0\_CELL\_VRAM[]

Bg1Index = BG1\_MAP\_VRAM[] -> BG1\_CELL\_VRAM[]

Bg2Index = BG2\_MAP\_VRAM[] -> BG2\_CELL\_VRAM[]

bgIndex = Bg0Index | Bg1Index | Bg2Index (first non-zero, by priority)

bgPaletteId = the paletteId for the winning BG

SCANLINE\_BUFFER\_VRAM[x] = PALETTE\_VRAM [bgPaletteId]

##### Find Visible Sprites

For(s=0;s<256;s++)

spriteData = SPRITE\_REGS\_VRAM[s]

If IsOnCurrentScanline(spriteData.Y)

Fifo.Push(spriteData )

##### Render Visible Sprites

Always…

// Spin through each X coordinate of a sprite (0..15)

spriteRenderX = spriteRenderX + 1;

// if on the last pixel, setup to fetch next sprite to render.

if(spriteRenderX == 16 - 1 )

fetchNextSpriteToRender = 1;

else

fetchNextSpriteToRender = 0;

// if now is a good time, then pop the next sprite to render.

If( !fifo.Empty() && fetchNextSpriteToRender )

spriteData = fifo.Pop();

hasSpriteToRender = true

// render the currently popped sprite.

if( hasSpriteToRender )

spriteImage = SPRITE\_REGS\_VRAM.Image;

spriteImageAddress = &SPRITE\_IMAGE\_VRAM[x+y\*16]

spriteImageColorIndex = \*spriteImageAddress;

spritePixelColor = PALETTE\_VRAM[SPRITE\_AREA | sprite.Palette \* 16 + spriteImageColorIndex]

SCANLINE\_BUFFER\_VRAM[spriteData.X + spriteRenderX] = spritePixelColor

// if it’s the last cycle of the scanline, then reset the sprite rendering engine.

fifo.reset()

hasSpriteToRender = false

##### Layers

Sprites

BG0 (good for fonts)

BG1 (foreground gameplay)

BG2 (background gameplay)

Background (Palette 0, ColorIndex 0)

## SDRAM

The SDRAM has the following attributes:

* 4MB of memory x 16bits = 8MB
* 1 Cycle sequential access @ the CPU’s clock speed of 72Mhz.
* 6 cycle penalty when changing pages (page size is 64 bytes).
* Periodic refresh of a row that cost 13 cycles.
* A full refresh of the entire device that costs about .8ms @ 24Mhz (thousands of cycles, or about 15 scanlines), has been observed to occur upon the first device access.
* Single Ported.

# Gremlin Memory Map

E000..FFFF 8K of Banked Memory (8 banks total, including VRAM, Device Registers, and 24K of extra SDRAM in 8K blocks).  
**NOTE: When power is applied, the Sprite Images VRAM is visible by default (bank 0), and CPU execution begins at E000 (i.e. the bootstrap code is placed in the Xilinx bytestream as sprite image data).**

0200..DFFF 56832 (55.5K) bytes of Main SDRAM.

0100..01FF 256 bytes of CPU Stack in SDRAM.

0008..00FF 248 bytes of Zero Page SDRAM.

0005..0007 NMI\_VECTOR. CPU jumps to this instruction whenever an external interrupt occurs.

0002..0004 BREAK\_VECTOR. CPU jumps to this location whenever the BRK instruction is executed.

0001..0001 1 Byte for a “Debug” Register. Bytes written here are sent to the DebugPinsManager, where they may be placed on external device pins, so that tools such as a logic analyzer can see what’s going on.

0000..0000 1 Byte for a “Bank Selection” register.

|  |  |  |
| --- | --- | --- |
| **Bank** | **Address Range** | **Description** |
| 0 | E000..FFFF | Sprite Images, and BOOT-STRAP code. |
| 1 | E000..EFFF | 4K of BG0 Cell Data |
| F000..FFFF | 4K of BG1 Cell Data |
| 2 | E000..EFFF | 4K of BG2 Cell Data |
| F000..F7FF | 2K of Sprite Registers |
| F800..FFFF | (not used – mirror of sprite regs) |
| 3 | E000..E7FF | 2K of BG0 MAP |
| E800..EFFF | 2K of BG1 MAP |
| F000..F7FF | 2K of BG2 MAP |
| F800..FFFF | 2K of Palettes |
| 4 | E000..E3FF | AUDIO Registers |
| E400..E7FF | VIDEO Registers |
| E800..EBFF | MMC Registers |
| EC00..EFFF | DMA Registers |
| F000..F3FF | KEYBOARD Registers |
| F400..F7FF | MOUSE Registers |
| F800..FBFF | BLITTER Registers |
| FC00..FFFF | COPPER Registers |
| 5 | E000..FFFF | 8K of SDRAM Bank 0 |
| 6 | E000..FFFF | 8K of SDRAM Bank 1 |
| 7 | E000..FFFF | 8K of SDRAM Bank 2 |

# Device Registers

## BG0\_MAP\_ADDRESS (no bits)

## BG0\_IMAGE\_ADDRESS (no bits)

## BG0\_GLOBAL\_PALETTE P

P = the 1st bit of the palette address for BG0. NOTE: All other bits of palette selection for BG0 are 0.

## BG0\_VISIBLE V

V = Visible (1=yes, 0=no)

## BG0\_SCROLL YYYXXX

X = 0..7, X smooth scroll value

Y = 0..7, Y smooth scroll value

## BG1\_MAP\_ADDRESS (no bits)

## BG1\_IMAGE\_ADDRESS (no bits)

## BG1\_GLOBAL\_PALETTE P

P = the 3rd bit of the palette address for BG1. NOTE: The 2nd bit of the palette address comes from each BG1 cell itself, meaning there is per-cell palette selection available. All other bits of palette selection for BG1 are 0.

## BG1\_VISIBLE V

V = Visible (1=yes, 0=no)

## BG1\_SCROLL YYYXXX

X = 0..7, X smooth scroll value

Y = 0..7, Y smooth scroll value

## BG2\_MAP\_ADDRESS (no bits)

## BG2\_IMAGE\_ADDRESS (no bits)

## BG2\_GLOBAL\_PALETTE P

P = the 5th bit of the palette address for BG2. NOTE: The 4th bit of the palette address comes from each BG2 cell itself, meaning there is per-cell palette selection available. All other bits of palette selection for BG2 are 0.

## BG2\_VISIBLE V

V = Visible (1=yes, 0=no)

## BG2\_SCROLL YYYXXX

X = 0..7, X smooth scroll value

Y = 0..7, Y smooth scroll value

## SPRITE\_DATA\_REGISTER[256] IIIIIIPP PPPPRRRY YYYYYYYX XXXXXXXX

I = ImageId (0..63)

P = PaletteId (0..63)

R = Rotation (0..7)

X = XPOS (0..511)

Y = YPOS (0..255)

## SPRITE\_POOL\_SELECT S

S = Sprite Pool Select(0 = sprites 0..255 are used, 1 = sprites 256..511 are used)